Version History REST API
Version: 1.0.0

OpenAPI2LaTeX Generator

December 19, 2022

Contents

Chapter 1

Information

REST API for common actions around document version history stored in **MongoDB** via mongo-service.

1.1 Formats

Document payloads can be encoded as either **BSON** or **JSON**. This applies to both input document payloads (CRUD API), and for the the outputs for all API endpoints.

JSON encoding is via the Mongo C driver utility function to convert **BSON** to **JSON**.

1.2 Compression

Compressed output is supported and encouraged. Only `gzip` compression is supported by the service. Output is compressed only if the output document size exceeds `128` bytes.

1.3 Correlation Id

Clients can specify a `correlationId` for each request if they wish to correlate log records that are generated across the system. This can be specified via a custom `x-spt-correlation-id` HTTP header.

1.4 Other Information

Version	1.0.0

Table 1.1: API Information

Local Instance	`http://localhost:6106`

Table 1.2: Server Information

Part I

Endpoints

Chapter 2

VersionHistory

2.1 list

Endpoint to retrieve metadata about history documents for specified entity.

Resource Path	/version/history/list/{database}/{collection}/{objectId}
HTTP Method	GET
Local Instance	http://localhost:6106

Returns metadata in chronological order of all versions for the specified entity.

2.1.1 Parameters

- **database** The database in which the entity is stored.

  in - path

  required - true

  schema

    type - string

    example - `sptdb`

- **collection** The collection in which the entity is stored.

  in - path

  required - true

  schema

    type - string

    example - `users`

- **objectId** The BSON object id for the entity.

in - path

required - true

schema

type - string

example - `5f3bc9e2502422053e08f9f1`

2.1.2 Responses

See table 2.1 for response codes and data.

Table 2.1: Responses for list

Code	Content Type	Notes
200	application/json	Document with a list of *version history* summary documents. **HistorySummary**. See chapter 9 on 40 for schema.
200	application/bson	Document with a list of *version history* summary documents. **HistorySummary**. See chapter 9 on 40 for schema.
400	application/json	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
404	application/json	If the specified `objectId` does not exist in the `database:collection`. **Error**. See chapter 7 on 35 for schema.
404	application/bson	If the specified `objectId` does not exist in the `database:collection`. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while retrieving the history document summary. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while retrieving the history document summary. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

continued on next page

continued from previous page

500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 24.1 on 70

2.2 document

Endpoint to retrieve the full version history document.

Resource Path	/version/history/document/{objectId}
HTTP Method	GET
Local Instance	http://localhost:6106

Returns the full version history document identified by the specified BSON object id.

2.2.1 Parameters

- **objectId** The BSON object id for the version history document. This is usually derived from a call to the `list` endpoint.

  in - path

  required - true

  schema

    type - string

    example - `5f3bc9e2502422053e08f9f1`

2.2.2 Responses

See table 2.2 for response codes and data.

Table 2.2: Responses for document

Code	Content Type	Notes
200	application/json	The complete version history document. The document versioned is available as the `entity` sub-document. **HistoryDocument**. See chapter 8 on 36 for schema.
200	application/bson	The complete version history document. The document versioned is available as the `entity` sub-document. **HistoryDocument**. See chapter 8 on 36 for schema.
400	application/json	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
404	application/json	If the specified `objectId` does not exist in the version history `database:collection`. **Error**. See chapter 7 on 35 for schema.
404	application/bson	If the specified `objectId` does not exist in the version history `database:collection`. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while retrieving the history document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while retrieving the history document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 25.1 on 71

2.3 entity

Endpoint to retrieve the original document stored in version history.

Resource Path	/version/history/entity/{objectId}
HTTP Method	GET
Local Instance	http://localhost:6106

Returns the **entity** that was versioned. Returns only the nested document without the version history wrapper.

2.3.1 Parameters

- **objectId** The BSON object id for the version history document. This is usually derived from a call to the `list` endpoint.

  in - path

  required - true

  schema

  type - string

  example - `5f3bc9e2502422053e08f9f1`

2.3.2 Responses

See table 2.3 for response codes and data.

Table 2.3: Responses for entity

Code	Content Type	Notes
200	application/json	The entity that was versioned. • **_id** BSON Object ID for the history document. – **Type** string – **Example** `5f3bc9e2502422053e08f9f1`

continued on next page

continued from previous page		
200	application/bson	The entity that was versioned. • **_id** BSON Object ID for the history document. – **Type** string – **Example** `5f3bc9e2502422053e08f9f1`
400	application/json	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.1 on 72

2.4 revert

Endpoint to revert the specified document to its previous version.

Resource Path	/version/history/revert/{historyObjectId}/{database}/{collection}/{entityObjectId}
HTTP Method	PUT
Local Instance	http://localhost:6106

Replaces the specified entity with its previous version specified by `historyObjectId`.

Note: This is a `PUT` operation as it modifies data in the database. No **payload** is expected from the client.

2.4.1 Parameters

- **historyObjectId** The BSON object id of the version history document to revert the entity to.

  in - path

  required - true

  schema

  type - string

  example - `5fa9910b8a64e17d2911e67a`

- **database** The database in which the entity is stored.

in - path

required - true

schema

type - string

example - `sptdb`

- **collection** The collection in which the entity is stored.

  in - path

  required - true

  schema

  type - string

  example - `users`

- **entityObjectId** The BSON object id for the entity to be reverted.

  in - path

  required - true

  schema

  type - string

  example - `5f3bc9e2502422053e08f9f1`

2.4.2 Responses

See table 2.4 for response codes and data.

Table 2.4: Responses for revert

Code	Content Type	Notes
200	application/json	The entity that was reverted. • **_id** BSON Object ID for the history document. – **Type** string – **Example** `5f3bc9e2502422053e08f9f1`

continued on next page

continued from previous page

200	application/bson	The entity that was reverted. • **_id** BSON Object ID for the history document. – **Type** string – **Example** `5f3bc9e2502422053e08f9f1`
400	application/json	If either of the ids specified is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If either of the ids specified is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
404	application/json	If the specified `historyObjectId` does not exist in the version history database collection. **Error**. See chapter 7 on 35 for schema.
404	application/bson	If the specified `historyObjectId` does not exist in the version history database collection. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while retrieving the history document summary. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while retrieving the history document summary. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.2 on 72

Chapter 3

CRUD

3.1 create

Endpoint to create a new document.

Resource Path	/crud/create/{database}/{collection}
HTTP Method	POST
Local Instance	http://localhost:6106

Create a new document as specified in the request payload in the `database:collection` specified as path parameters.

The response will be the small metadata document as returned by the `mongo-service`.

Note: This is a `POST` operation as the process is not idempotent. Only the first request to create a document with a specified object id will succeed. Any subsequent requests with the same payload will fail.

3.1.1 Parameters

- **database** The database in which the entity is to be created.

  in - path

  required - true

  schema

    type - string

    example - `sptdb`

- **collection** The collection in which the entity is to be created.

  in - path

required - true

schema

type - string

example - `users`

3.1.2 Request Body

Table 3.1: Request body for create

Property	Value
Required	true
Content	**application/json** **CreateDocument**. See chapter 4 on 30 **application/bson** **CreateDocument**. See chapter 4 on 30

3.1.3 Responses

See table 3.2 for response codes and data.

Table 3.2: Responses for create

Code	Content Type	Notes
200	application/json	The entity that was created. **CreateResponse**. See chapter 5 on 31 for schema.
200	application/bson	The entity that was created. **CreateResponse**. See chapter 5 on 31 for schema.
400	application/json	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while creating the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while creating the document. **Error**. See chapter 7 on 35 for schema.

continued on next page

continued from previous page

500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.3 on 73

3.2 update

Endpoint to modify existing document(s) with the input data.

Resource Path	/crud/update/{database}/{collection}
HTTP Method	POST
Local Instance	http://localhost:6106

Merges all documents in the specified `database:collection` matching the filter query with the specified document.

Version history documents with the post-update version of the matching documents are created.

Note: This is a `POST` request since the number of documents updated can vary between different invocations of this endpoint depending upon the `filter` query.

3.2.1 Parameters

- **database** The database in which the entity exists.

  in - path

  required - true

  schema

  type - string
  example - `sptdb`

- **collection** The collection in which the entity exists.

  in - path

  required - true

  schema

type - string
example - `users`

3.2.2 Request Body

Table 3.3: Request body for update

Property	Value
Required	true
Content	**application/json** **UpdateRequest**. See chapter 13 on 47 **application/bson** **UpdateRequest**. See chapter 13 on 47

3.2.3 Responses

See table 3.4 for response codes and data.

Table 3.4: Responses for update

Code	Content Type	Notes
200	application/json	A document with basic metadata about the results of the update operation. **UpdatesResponse**. See chapter 15 on 51 for schema.
200	application/bson	A document with basic metadata about the results of the update operation. **UpdatesResponse**. See chapter 15 on 51 for schema.
400	application/json	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while updating documents. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while updating documents. **Error**. See chapter 7 on 35 for schema.

continued on next page

continued from previous page		
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.4 on 74

3.3 updateById

Endpoint to replace an existing document with the input data.

Resource Path	/crud/update/{database}/{collection}/{id}
HTTP Method	PUT
Local Instance	http://localhost:6106

Replaces the document identified by the specified `database:collection:id` with the input payload.

A version history document with the payload contents is created.

The input payload **must** contain the `_id` property.

3.3.1 Parameters

- **database** The database in which the entity exists.

  in - path

  required - true

  schema

  type - string

  example - `sptdb`

- **collection** The collection in which the entity exists.

  in - path

  required - true

  schema

  type - string

example - `users`

- **id** The BSON object id for the document to be updated.

  in - path

  required - true

  schema

    type - string

    example - `5f3bc9e2502422053e08f9f1`

3.3.2 Request Body

Table 3.5: Request body for updateById

Property	Value
Required	true
Content	**application/json** **CreateDocument**. See chapter 4 on 30 **application/bson** **CreateDocument**. See chapter 4 on 30

3.3.3 Responses

See table 3.6 for response codes and data.

Table 3.6: Responses for updateById

Code	Content Type	Notes
200	application/json	A document that contains the updated entity as well as information about the version history that was created. **UpdateResponse**. See chapter 14 on 49 for schema.
200	application/bson	A document that contains the updated entity as well as information about the version history that was created. **UpdateResponse**. See chapter 14 on 49 for schema.

continued on next page

continued from previous page		
400	application/json	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while updating the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while updating the document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.5 on 75

3.4 mergeById

Endpoint to merge the input data into the specified document.

Resource Path	/crud/update/{database}/{collection}/{id}
HTTP Method	PATCH
Local Instance	http://localhost:6106

Merges the payload data into the document identified by the specified `database:collection:id`.

A version history document with the post-update contents of the document is created.

The input payload **must** contain the `_id` property.

3.4.1 Parameters

- **database** The database in which the entity exists.

  in - path

  required - true

  schema

type - string
example - `sptdb`

- **collection** The collection in which the entity exists.

  in - path

  required - true

  schema

  type - string
  example - `users`

- **id** The BSON object id for the document to be updated.

  in - path

  required - true

  schema

  type - string
  example - `5f3bc9e2502422053e08f9f1`

3.4.2 Request Body

Table 3.7: Request body for mergeById

Property	**Value**
Required	true
Content	**application/json** **UpdateResponse**. See chapter 14 on 49 **application/bson** **UpdateResponse**. See chapter 14 on 49

3.4.3 Responses

See table 3.8 for response codes and data.

Table 3.8: Responses for mergeById

Code	Content Type	Notes
200	application/json	A document that contains the updated entity as well as information about the version history that was created. **UpdateResponse**. See chapter 14 on 49 for schema.
200	application/bson	A document that contains the updated entity as well as information about the version history that was created. **UpdateResponse**. See chapter 14 on 49 for schema.
400	application/json	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while updating the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while updating the document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.6 on 76

3.5 replace

Endpoint to replace a document matched by a specified filter query.

Resource Path	/crud/replace/{database}/{collection}
HTTP Method	POST
Local Instance	http://localhost:6106

Replace an existing document matched by an input `filter` query document with the specified `replace` document.

If multiple documents match the `filter`, the first one returned by MongoDB is replaced.

The replacement document need not include the `_id property. If the caller has access to the _id`, there would be no reason to use a `filter` query. The same can be achieved by making a `PUT` request to the *update* endpoint. Of course the `filter` query can be by `_id` property.

3.5.1 Parameters

- **database** The database in which the entity exists.

  in - path

  required - true

  schema

    type - string

    example - `sptdb`

- **collection** The collection in which the entity exists.

  in - path

  required - true

  schema

    type - string

    example - `users`

3.5.2 Request Body

Table 3.9: Request body for replace

Property	**Value**
Required	true
Content	**application/json** **ReplaceRequest**. See chapter 11 on 43 **application/bson** **ReplaceRequest**. See chapter 11 on 43

3.5.3 Responses

See table 3.10 for response codes and data.

Table 3.10: Responses for replace

Code	Content Type	Notes
200	application/json	A document that contains the updated entity as well as information about the version history that was created. **UpdateResponse**. See chapter 14 on 49 for schema.
200	application/bson	A document that contains the updated entity as well as information about the version history that was created. **UpdateResponse**. See chapter 14 on 49 for schema.
400	application/json	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the input document payload is not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while updating the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while updating the document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 26.7 on 77

3.6 retrieve

Endpoint to retrieve documents with the specified combination.

Resource Path	/crud/retrieve/{database}/{collection}/{property}/{value}
HTTP Method	GET
Local Instance	http://localhost:6106

Retrieve document(s) from the specified `database:collection` matching the specified combination of property name and value.

Since the `value` is specified as a `path` parameter, this is intended for use for simple `string` values which do not have spaces or other special characters. When specifying complex values, the request `path` must be properly *URL encoded*.

The `property` **must** hold a `string` type value for this endpoint to work. The only exception is the case for *id*, `in which case the value will be parsed as a BSON Object Id. This also means that non object id id` values are not supported.

The response will be the entire document(s) as stored in the specified database collection.

If the `property` specified is the `_id` property the response will include only the `result` object. For any other property (even if it is a *unique* property) the response will include the `results` array. The response will have only one or the other of `result` or `results` (mutually exclusive).

3.6.1 Parameters

- **database** The database from which entities are to be retrieved.

  in - path

  required - true

  schema

  type - string

  example - `sptdb`

- **collection** The collection from which entities are to be retrieved.

  in - path

  required - true

  schema

  type - string

  example - `users`

- **property** The name of the property to find matching document.

  in - path

  required - true

  schema

  type - string

  example - `email`

- **value** The value to match for the specified property.

  in - path

  required - true

  schema

  type - string

  example - `test@test.com`

3.6.2 Responses

See table 3.11 for response codes and data.

Table 3.11: Responses for retrieve

Code	Content Type	Notes
200	application/json	The document(s) that were retrieved. **RetrieveResponse**. See chapter 12 on 45 for schema.
200	application/bson	The document(s) that were retrieved. **RetrieveResponse**. See chapter 12 on 45 for schema.
400	application/json	If the path parameters are not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the path parameters are not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while creating the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while creating the document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 27.1 on 78

3.7 query

Endpoint to retrieve documents matching a query.

Resource Path	/crud/query/{database}/{collection}
HTTP Method	POST
Local Instance	http://localhost:6106

Execute the specified query against the specified `database:collection`.

If the `query` specified is the `_id` property the response will include only the `result` object. For any other query the response will include the `results` array. The response will have only one or the other of `result` or `results` (mutually exclusive).

Note: If no `options` is specified a default `options` with a `limit` of `100` is applied. This is to avoid running the instance out of memory unless the caller explicitly requests a larger result set. When the caller explicitly specified `options`, the expectation is that the caller has also specified a *sane* `limit`.

3.7.1 Parameters

- **database** The database from which entities are to be retrieved.

  in - path

  required - true

  schema

    type - string

    example - `sptdb`

- **collection** The collection from which entities are to be retrieved.

  in - path

  required - true

  schema

    type - string

    example - `users`

3.7.2 Request Body

Table 3.12: Request body for query

Property	**Value**
Required	true
Content	**application/json** **QueryDocument**. See chapter 10 on 41 **application/bson** **QueryDocument**. See chapter 10 on 41

continued on next page

continued from previous page

3.7.3 Responses

See table 3.13 for response codes and data.

Table 3.13: Responses for query

Code	Content Type	Notes
200	application/json	The document(s) that were retrieved. **RetrieveResponse**. See chapter 12 on 45 for schema.
200	application/bson	The document(s) that were retrieved. **RetrieveResponse**. See chapter 12 on 45 for schema.
400	application/json	If the path parameters are not valid. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If the path parameters are not valid. **Error**. See chapter 7 on 35 for schema.
412	application/json	If the input query specification is not valid. **Error**. See chapter 7 on 35 for schema.
412	application/bson	If the input query specification is not valid. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while creating the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while creating the document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Code Samples

See section 27.2 on 79

3.8 delete

Endpoint to delete a document.

Resource Path	/crud/delete/{database}/{collection}/{id}
HTTP Method	DELETE
Local Instance	http://localhost:6106

Delete the specified document from the database. Note that deleting documents from the version history `database:collection` is not supported.

3.8.1 Parameters

- **database** The database from which the entity is to be deleted.

  in - path

  required - true

  schema

  type - string

  example - `sptdb`

- **collection** The collection from which the entity is to be deleted.

  in - path

  required - true

  schema

  type - string

  example - `users`

- **id** The BSON object id for the document to be deleted.

  in - path

  required - true

  schema

  type - string

  example - `5f3bc9e2502422053e08f9f1`

3.8.2 Responses

See table 3.14 for response codes and data.

Table 3.14: Responses for delete

Code	Content Type	Notes
200	application/json	Document with information about the document that was deleted. Of most interest is the information returned about the version history document that was created as a result of the `delete`. **DeleteResponse**. See chapter 6 on 33 for schema.
200	application/bson	Document with information about the document that was deleted. Of most interest is the information returned about the version history document that was created as a result of the `delete`. **DeleteResponse**. See chapter 6 on 33 for schema.
400	application/json	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
400	application/bson	If `objectId` is not a valid BSON object id. **Error**. See chapter 7 on 35 for schema.
404	application/json	If the specified `objectId` does not exist in the specified `database:collection`. **Error**. See chapter 7 on 35 for schema.
404	application/bson	If the specified `objectId` does not exist in the specified `database:collection`. **Error**. See chapter 7 on 35 for schema.
417	application/json	If the `mongo-service` returns an error while deleting the document. **Error**. See chapter 7 on 35 for schema.
417	application/bson	If the `mongo-service` returns an error while deleting the document. **Error**. See chapter 7 on 35 for schema.
500	application/json	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.
500	application/bson	If errors were encountered communicating with `mongo-service`. **Error**. See chapter 7 on 35 for schema.

Part II

Schemas

Chapter 4

CreateDocument

Standard structure for a payload document sent to the utility `create` endpoint.

Only the mandatory `_id` property is documented. Other properties can be included as appropriate, and will be saved to the specified `database:collection`.

4.1 _id

BSON Object ID for the document to be created.

Note: When sending data in **JSON** format, make sure it is wrapped into an object (`"_id": "$oid": "5f3bc9e29ba4f45f810edf22"`) to be parsed into **BSON** using MongoDB C driver JSON parser.

Table 4.1: Properties for ::CreateDocument::_id

Property	**Value**
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

Chapter 5

CreateResponse

Structure returned by the `create` document endpoint. This is the same structure that is returned by the `mongo-service`. Only minimal metadata is returned to the client.

5.1 _id

BSON Object ID for the document that was created. This is the same value that was specified in the request.

Note: When response data is in **JSON** format, the *`id`* `is wrapped in an object ("id": "$oid": "5f3bc9e29ba4f45f810edf22" )` by the MongoDB C driver JSON converter.

Table 5.1: Properties for ::CreateResponse::_id

Property	**Value**
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

5.2 database

The *version history* database in which the initial `create` version of the document was stored.

In keeping with the version history implementation, the input document is duplicated in the version history database.

Table 5.2: Properties for ::CreateResponse::database

Property	**Value**
Type	string
Required	true
Example	versionHistory

5.3 collection

The *version history* collection in which the initial `create` version of the document was stored.

Table 5.3: Properties for ::CreateResponse::collection

Property	**Value**
Type	string
Required	true
Example	entities

5.4 entity

BSON Object ID for the version history document that was created. This can be used to retrieve the history document.

Note: When response data is in **JSON** format, the *`id`* `is wrapped in an object ("id":    "$oid": "5f35e5e19e48c37186539141" )` by the MongoDB C driver JSON converter.

Table 5.4: Properties for ::CreateResponse::entity

Property	**Value**
Type	string
Required	true
Example	5f35e5e19e48c37186539141

Chapter 6

DeleteResponse

Structure returned by the `delete` document endpoint. This is a simplified representation of the structure that is returned by the `mongo-service`. In particular, the arrays returned by the service are transformed into singular objects since this endpoint only allows deleting a single document.

6.1 _id

BSON Object ID for the document that was deleted. This is the same value that was specified in the path.

Note: When response data is in **JSON** format, the `id is wrapped in an object ("id": "$oid": "5f3bc9e29ba4f45f810edf22" )` by the MongoDB C driver JSON converter.

Table 6.1: Properties for ::DeleteResponse::_id

Property	Value
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

6.2 history

Metadata about the version history document that was created as a result of deleting the specified document. Clients may use this information to *revert* or *undo* the operation if so desired.

Table 6.2: Properties for ::DeleteResponse::history

Property	**Value**
Type	object
Required	true

Chapter 7

Error

Common format for returning errors from the service.

7.1 code

Usually the same as the HTTP status code for the response.

Table 7.1: Properties for ::Error::code

Property	**Value**
Type	integer
Required	true

7.2 cause

A short description of the cause of error.

Table 7.2: Properties for ::Error::cause

Property	**Value**
Type	string
Required	true
Example	Not found

Chapter 8

HistoryDocument

Standard structure for version history document.

8.1 _id

BSON Object ID for the history document.

Table 8.1: Properties for ::HistoryDocument::_id

Property	Value
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

8.2 database

Database in which the versioned entity is (or was) stored.

Table 8.2: Properties for ::HistoryDocument::database

Property	Value
Type	string
Required	true
Example	sptdb

8.3 collection

Collection in which the versioned entity is (or was) stored.

Table 8.3: Properties for ::HistoryDocument::collection

Property	**Value**
Type	string
Required	true
Example	users

8.4 action

The `action` which resulted in the history document being created.

Table 8.4: Properties for ::HistoryDocument::action

Property	**Value**
Type	string
Required	true
Example	update
Enum	Allowed values - `create,update,delete`

8.5 created

The timestamp at which the version was created.

Table 8.5: Properties for ::HistoryDocument::created

Property	**Value**
Type	string
Required	true
Format	date-time

8.6 entity

The document that was versioned. At the very least it will contain an `_id` property as documented here. The document will have other properties as was saved originally.

Table 8.6: Properties for ::HistoryDocument::entity

Property	Value
Type	object
Required	true

8.7 metadata

Optional metadata associted with the version. This is user supplied information when submitting payloads to `mongo-service`.

Table 8.7: Properties for ::HistoryDocument::metadata

Property	Value
Type	object
Required	false

Chapter 9

HistorySummary

Version history summary information.

9.1 results

Table 9.1: Properties for ::HistorySummary::results

Property	**Value**
Type	array
Required	true

Chapter 10

QueryDocument

A MongoDB query document.

When making the query using JSON encoding, ensure that the query can be parsed into BSON using the MongoDB driver.

10.1 query

The query will be passed *as-is* to the `mongo-service`. No properties are documented as there are no mandatory properties. Consult the MongoDB documentation for valid query syntax.

Table 10.1: Properties for ::QueryDocument::query

Property	**Value**
Type	object
Required	true

Code Example

```
{
  "group": "admin",
  "created": {
    "$gt": {
      "$date": "2020-08-18T12:30:00.000Z"
    }
  }
}
```

10.2 options

Options to control output from the *query*. Use to control result sorting, properties included in the returned documents etc.

Note: Callers are strongly adviced to *always* specify the `limit` option to restrict the maximum number of matching documents. Use standard *pagination* techniques (include a `$gt` clause when retrieving additional results) to ensure the query is optimal.

Table 10.2: Properties for ::QueryDocument::options

Property	**Value**
Type	object
Required	false

Code Example

```
{
  "limit": 100,
  "sort": {
    "_id": -1
  },
  "projection": {
    "name": 1,
    "email": 1
  }
}
```

Chapter 11

ReplaceRequest

Structure for a general purpose replace request. Replace is expressed as a combination of an update `filter` query (should return a single matching document), and the `replace` document to replace the existing document in the specified `database:collection`.

Since this is a full replace, the replacement document must be the full document (the `_id` field is optional).

The post-update document is retrieved (if `_id` is not included) to create the version history document.

11.1 filter

The filter query to use to find the candidate document that is to be updated. If the filter matches multiple documents, the first one returned is updated.

Note: When sending data in **JSON** format, make sure special types such as BSON Object Id, date, etc are encoded in the format expected by the MongoDB JSON parser.

Table 11.1: Properties for ::ReplaceRequest::filter

Property	**Value**
Type	object
Required	true

Code Example

```json
{
  "group": "users",
  "role": "admin"
}
```

11.2 replace

The document to replace the existing document that matches the filter query.

Note: The *id* `is not required. MongoDB handles merging the existing id` with the `replace` document as needed.

Table 11.2: Properties for ::ReplaceRequest::replace

Property	**Value**
Type	object
Required	true

Code Example

```
{
  "_id": {
    "$oid": "5f35e887e799c5218603915b"
  },
  "modified": {
    "$date": 1608780307513
  },
  "user": {
    "_id": {
      "$oid": "5fe409f879634171a83232a3"
    },
    "username": "rakesh"
  }
}
```

Chapter 12

RetrieveResponse

General description of the document that is returned by the `mongo-service` when retrieving documents.

12.1 result

A single document that is returned when the retrieval was performed by the BSON ObjectId `_id` property.

Table 12.1: Properties for ::RetrieveResponse::result

Property	**Value**
Type	object
Required	false

Code Example

```
{
  "_id": {
    "$oid": "5f35e5e1e799c52186039122"
  },
  "intValue": 123,
  "floatValue": 123.0,
  "boolValue": true,
  "stringValue": "abc123",
  "nested": {
    "key": "value"
  }
}
```

12.2 results

Array of matching documents returned when the retrieval was performed by any other property (or arbitrarily complex query). If the match was performed on a `unique` property, the array will have only one document.

Table 12.2: Properties for ::RetrieveResponse::results

Property	**Value**
Type	array
Required	false

Chapter 13

UpdateRequest

Structure for a general purpose update request. Update is expressed as a combination of an update `filter` query, and the document to merge in to all matching documents in the specified `database:collection`.

This endpoint can be used to apply standard information to a number of documents matching the input filter. In the extreme case, this can be used to add some standard properties to all documents in a `collection`.

Each document matching the `filter` is retrieved post-merge, and a corresponding version history document generated. This post-processing can add quite a lot of time to the request.

Note: When updating documents by a filter query with `_id` property, use the endpoint that takes the `id` path parameter.

13.1 filter

The filter query to use to find candidate documents that are to be updated. If the filter matches multiple documents, all matching documents are updated.

Note: When sending data in **JSON** format, make sure special types such as BSON Object Id, date, etc are encoded in the format expected by the MongoDB JSON parser.

Table 13.1: Properties for ::UpdateRequest::filter

Property	Value
Type	object
Required	true
continued on next page	

continued from previous page

Code Example

```
{
  "group": "users",
  "role": "admin"
}
```

13.2 update

The data that is to be merged into all documents matching the input filter query. In general this should not include the `_id` (or any unique fields for that matter) field, unless the input filter query matches exactly one document.

Table 13.2: Properties for ::UpdateRequest::update

Property	**Value**
Type	object
Required	true

Code Example

```
{
  "modified": {
    "$date": 1608780307513
  },
  "user": {
    "_id": {
      "$oid": "5fe409f879634171a83232a3"
    },
    "username": "rakesh"
  }
}
```

Chapter 14

UpdateResponse

Structure returned by the `update` document endpoint. This is the same structure that is returned by the `mongo-service`.

The `document` element in the response will be entire current document in the `database:collection`. Only the mandatory `_id` can be documented.

14.1 document

The full (result after merging input) document that is the current content for the specified document.

Table 14.1: Properties for ::UpdateResponse::document

Property	**Value**
Type	object
Required	false

Code Example

```
{
  "_id": {
    "$oid": "5f35e5e1e799c52186039122"
  },
  "intValue": 123,
  "floatValue": 123.0,
  "boolValue": true,
  "stringValue": "abc123",
  "nested": {
    "key": "value"
  }
}
```

14.2 history

Metadata about the version history document that was generated by the update action.

Table 14.2: Properties for ::UpdateResponse::history

Property	Value
Type	object
Required	false

Code Example

```
{
  "_id": {
    "$oid": "5f35e887e799c5218603915b"
  },
  "database": "sptdb",
  "collection": "users",
  "entity": {
    "$oid": "5f35e5e1e799c52186039122"
  }
}
```

Chapter 15

UpdatesResponse

Response returned in response to multi-document update by filter query. Only very minimal information is returned in the response.

15.1 success

The number of successful updates for the input filter.

Table 15.1: Properties for ::UpdatesResponse::success

Property	**Value**
Type	integer
Required	false

15.2 failure

The number of failed updates for the input filter.

Table 15.2: Properties for ::UpdatesResponse::failure

Property	**Value**
Type	integer
Required	false

15.3 history

Array of BSON ObjectId values for the version history documents created for the updates.

Table 15.3: Properties for ::UpdatesResponse::history

Property	**Value**
Type	array
Required	false

Chapter 16

CreateResponse

Structure returned by the `create` document endpoint. This is the same structure that is returned by the `mongo-service`. Only minimal metadata is returned to the client.

16.1 _id

BSON Object ID for the document that was created. This is the same value that was specified in the request.

Note: When response data is in **JSON** format, the *id* `is wrapped in an object ("id": "$oid": "5f3bc9e29ba4f45f810edf22" )` by the MongoDB C driver JSON converter.

Table 16.1: Properties for openapi::CreateResponse::_id

Property	**Value**
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

16.2 database

The *version history* database in which the initial `create` version of the document was stored.

In keeping with the version history implementation, the input document is duplicated in the version history database.

Table 16.2: Properties for openapi::CreateResponse::database

Property	Value
Type	string
Required	true
Example	versionHistory

16.3 collection

The *version history* collection in which the initial `create` version of the document was stored.

Table 16.3: Properties for openapi::CreateResponse::collection

Property	Value
Type	string
Required	true
Example	entities

16.4 entity

BSON Object ID for the version history document that was created. This can be used to retrieve the history document.

Note: When response data is in **JSON** format, the *id* `is wrapped in an object ("id": "$oid": "5f35e5e19e48c37186539141"`) by the MongoDB C driver JSON converter.

Table 16.4: Properties for openapi::CreateResponse::entity

Property	Value
Type	string
Required	true
Example	5f35e5e19e48c37186539141

Chapter 17

DeleteResponse

Structure returned by the `delete` document endpoint. This is a simplified representation of the structure that is returned by the `mongo-service`. In particular, the arrays returned by the service are transformed into singular objects since this endpoint only allows deleting a single document.

17.1 _id

BSON Object ID for the document that was deleted. This is the same value that was specified in the path.

Note: When response data is in **JSON** format, the *id* `is wrapped in an object ("id": "$oid": "5f3bc9e29ba4f45f810edf22" )` by the MongoDB C driver JSON converter.

Table 17.1: Properties for openapi::DeleteResponse::_id

Property	**Value**
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

17.2 history

Metadata about the version history document that was created as a result of deleting the specified document. Clients may use this information to *revert* or *undo* the operation if so desired.

Table 17.2: Properties for openapi::DeleteResponse::history

Property	**Value**
Type	object
Required	true

Chapter 18

Error

Common format for returning errors from the service.

18.1 code

Usually the same as the HTTP status code for the response.

Table 18.1: Properties for openapi::Error::code

Property	**Value**
Type	integer
Required	true

18.2 cause

A short description of the cause of error.

Table 18.2: Properties for openapi::Error::cause

Property	**Value**
Type	string
Required	true
Example	Not found

Chapter 19

HistoryDocument

Standard structure for version history document.

19.1 _id

BSON Object ID for the history document.

Table 19.1: Properties for openapi::HistoryDocument::_id

Property	Value
Type	string
Required	true
Example	5f3bc9e2502422053e08f9f1

19.2 database

Database in which the versioned entity is (or was) stored.

Table 19.2: Properties for openapi::HistoryDocument::database

Property	Value
Type	string
Required	true
Example	sptdb

19.3 collection

Collection in which the versioned entity is (or was) stored.

Table 19.3: Properties for openapi::HistoryDocument::collection

Property	Value
Type	string
Required	true
Example	users

19.4 action

The `action` which resulted in the history document being created.

Table 19.4: Properties for openapi::HistoryDocument::action

Property	Value
Type	string
Required	true
Example	update
Enum	Allowed values - `create,update,delete`

19.5 created

The timestamp at which the version was created.

Table 19.5: Properties for openapi::HistoryDocument::created

Property	Value
Type	string
Required	true
Format	date-time

19.6 entity

The document that was versioned. At the very least it will contain an `_id` property as documented here. The document will have other properties as was saved originally.

Table 19.6: Properties for openapi::HistoryDocument::entity

Property	Value
Type	object
Required	true

19.7 metadata

Optional metadata associted with the version. This is user supplied information when submitting payloads to `mongo-service`.

Table 19.7: Properties for openapi::HistoryDocument::metadata

Property	Value
Type	object
Required	false

Chapter 20

HistorySummary

Version history summary information.

20.1 results

Table 20.1: Properties for openapi::HistorySummary::results

Property	**Value**
Type	array
Required	true

Chapter 21

RetrieveResponse

General description of the document that is returned by the `mongo-service` when retrieving documents.

21.1 result

A single document that is returned when the retrieval was performed by the BSON ObjectId `_id` property.

Table 21.1: Properties for openapi::RetrieveResponse::result

Property	**Value**
Type	object
Required	false

Code Example

```
{
  "_id": {
    "$oid": "5f35e5e1e799c52186039122"
  },
  "intValue": 123,
  "floatValue": 123.0,
  "boolValue": true,
  "stringValue": "abc123",
  "nested": {
    "key": "value"
  }
}
```

21.2 results

Array of matching documents returned when the retrieval was performed by any other property (or arbitrarily complex query). If the match was performed on a `unique` property, the array will have only one document.

Table 21.2: Properties for openapi::RetrieveResponse::results

Property	**Value**
Type	array
Required	false

Chapter 22

UpdateResponse

Structure returned by the `update` document endpoint. This is the same structure that is returned by the `mongo-service`.

The `document` element in the response will be entire current document in the `database:collection`. Only the mandatory `_id` can be documented.

22.1 document

The full (result after merging input) document that is the current content for the specified document.

Table 22.1: Properties for openapi::UpdateResponse::document

Property	**Value**
Type	object
Required	false

Code Example

```
{
  "_id": {
    "$oid": "5f35e5e1e799c52186039122"
  },
  "intValue": 123,
  "floatValue": 123.0,
  "boolValue": true,
  "stringValue": "abc123",
  "nested": {
    "key": "value"
  }
}
```

22.2 history

Metadata about the version history document that was generated by the update action.

Table 22.2: Properties for openapi::UpdateResponse::history

Property	Value
Type	object
Required	false

Code Example

```
{
  "_id": {
    "$oid": "5f35e887e799c5218603915b"
  },
  "database": "sptdb",
  "collection": "users",
  "entity": {
    "$oid": "5f35e5e1e799c52186039122"
  }
}
```

Chapter 23

UpdatesResponse

Response returned in response to multi-document update by filter query. Only very minimal information is returned in the response.

23.1 success

The number of successful updates for the input filter.

Table 23.1: Properties for openapi::UpdatesResponse::success

Property	**Value**
Type	integer
Required	false

23.2 failure

The number of failed updates for the input filter.

Table 23.2: Properties for openapi::UpdatesResponse::failure

Property	**Value**
Type	integer
Required	false

23.3 history

Array of BSON ObjectId values for the version history documents created for the updates.

Table 23.3: Properties for openapi::UpdatesResponse::history

Property	**Value**
Type	array
Required	false

Part III

Code Samples

Chapter 24

VersionHistory

24.1 list

24.1.1 C++

```cpp
QString url{ "http://localhost:6106/version/history/list/" };
using ReplyPointer = std::unique_ptr<QNetworkReply>;
const auto get = []( const QString& url, QNetworkRequest* req )
{
  QEventLoop eventLoop;
  connect( &mgr, &QNetworkAccessManager::finished, &eventLoop, &QEventLoop::quit );

  req->setUrl( QUrl( url ) );
  req->setAttribute( QNetworkRequest::Http2DirectAttribute, {true} );
  ReplyPointer rptr{ mgr.get( *req ) };
  eventLoop.exec();
  return rptr ;
};

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

const auto endpoint = QString( "%1itest/test/%4" ).arg( url ).arg( entityId );
const auto reply = get( endpoint, &req );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error retrieving api response" )
    ;
const auto body = reply->readAll();

const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "results" ) != option->end(), "Matching entities not returned
    " );

QVERIFY2( (*option)["results"].type() == bsoncxx::type::k_array, "Results not array"
    );
```

Chapter 25

VersionHistory

25.1 document

25.1.1 C++

```cpp
QString url{ "http://localhost:6106/version/history/document/" };
QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

const auto endpoint = QString( "%1%2" ).arg( url ).arg( historyId );
const auto reply = get( endpoint, &req );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error retrieving api response" )
    ;
const auto body = reply->readAll();

const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "database" ) != option->end(), "database not returned" );
QVERIFY2( option->find( "collection" ) != option->end(), "database not returned" );
QVERIFY2( option->find( "action" ) != option->end(), "database not returned" );
QVERIFY2( option->find( "entity" ) != option->end(), "database not returned" );
```

Chapter 26

VersionHistory

26.1 entity

26.1.1 C++

```cpp
QString url{ "http://localhost:6106/version/history/entity/" };
QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

const auto endpoint = QString( "%1%2" ).arg( url ).arg( historyId );
const auto reply = get( endpoint, &req );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error retrieving api response" )
    ;
const auto body = reply->readAll();

const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "database" ) == option->end(), "wrapper database returned" );
QVERIFY2( option->find( "collection" ) == option->end(), "wrapper database returned"
    );
QVERIFY2( option->find( "action" ) == option->end(), "wrapper database returned" );
QVERIFY2( option->find( "entity" ) == option->end(), "wrapper database returned" );
```

26.2 revert

26.2.1 C++

```
QString url{ "http://localhost:6106/version/history/revert/" };
QNetworkRequest req;
req.setRawHeader( "accept", "application/json" );

const auto endpoint = QString( "%1%2/itest/test/%3" ).arg( url ).arg( historyId ).arg
    ( entityId );
const auto reply = get( endpoint, &req );

QVERIFY2( reply->error() != QNetworkReply::NoError, "PUT allowed on api endpoint" );
const auto doc = QJsonDocument::fromJson( reply->readAll() );
const auto obj = doc.object();
QVERIFY2( !obj.isEmpty(), "Empty error response for api endpoint" );
QVERIFY2( !obj["cause"].toString().isEmpty(), "Error response does not have cause" );
```

26.3 create

26.3.1 C++

```
QString url{ "http://localhost:6106/crud" };

const auto post = []( const QString& url, const QByteArray& data,
    QNetworkRequest* req, const QString& contentType )
{
  QEventLoop eventLoop;
  connect( &mgr, &QNetworkAccessManager::finished, &eventLoop, &QEventLoop::quit );

  req->setUrl( QUrl( url ) );
  req->setHeader( QNetworkRequest::ContentTypeHeader, contentType );
  req->setAttribute( QNetworkRequest::Http2DirectAttribute, {true} );
  ReplyPointer rptr{ mgr.post( *req, data ) };
  eventLoop.exec();
  return rptr ;
};

using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::open_document;
using bsoncxx::builder::stream::close_document;
using bsoncxx::builder::stream::finalize;

auto id = bsoncxx::oid{};
entityId2 = QString::fromStdString( id.to_string() );

auto doc = document{} <<
  "_id" << id <<
  "string" << "string value 2" <<
  "integer" << 2 <<
  "double" << 2.0 <<
  "boolean" << true <<
  "nested" <<
    open_document <<
      "key1" << "value2" <<
      "key2" << "value2" <<
    close_document <<
```

```
finalize;

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

const auto endpoint = QString( "%1/create/itest/test" ).arg( url );
const auto reply = post( endpoint,
    { reinterpret_cast<const char*>( doc.view().data() ), static_cast<int>( doc.view
    ().length() ) },
    &req, "application/bson" );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error creating document" );
const auto body = reply->readAll();
const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "_id" ) != option->end(), "Create response did not return id"
     );
```

26.4 update

26.4.1 C++

```
QString url{ "http://localhost:6106/crud/update/itest/test" };

using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::open_document;
using bsoncxx::builder::stream::close_document;
using bsoncxx::builder::stream::finalize;

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

auto bdoc = document{} <<
  "filter" << open_document << "prop1" << "value1" << close_document <<
  "update" << open_document << "added" << "another value" << close_document <<
  finalize;
const auto reqv = bdoc.view();
auto payload = QByteArray( reinterpret_cast<const char*>( reqv.data() ), reqv.length
    () );

const auto reply = post( url, payload, &req );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error updating document" );
const auto body = reply->readAll();
const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
qDebug() << QString::fromStdString( bsoncxx::to_json( *option ) );
QVERIFY2( option->find( "success" ) != option->end(), "Update response did not return
     updated document count" );
QVERIFY2( option->find( "history" ) != option->end(), "Update response did not return
     history metadata" );
```

```
QVERIFY2( option->find( "failure" ) != option->end(), "Update response did not
    include failure array" );

const auto arr = (*option)["failure"].get_array();
QVERIFY2( std::distance( arr.value.begin(), arr.value.end() ) == 0, "Update request
    had failures" );
```

26.5 updateById

26.5.1 C++

```
QString url{ "http://localhost:6106/crud/update/itest/test" };

const auto put = []( const QString& url, const QByteArray& data,
    QNetworkRequest* req, const QString& contentType )
{
  QEventLoop eventLoop;
  connect( &mgr, &QNetworkAccessManager::finished, &eventLoop, &QEventLoop::quit );

  req->setUrl( QUrl( url ) );
  req->setHeader( QNetworkRequest::ContentTypeHeader, contentType );
  req->setAttribute( QNetworkRequest::Http2DirectAttribute, {true} );
  ReplyPointer rptr{ mgr.put( *req, data ) };
  eventLoop.exec();
  return rptr ;
};

using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::finalize;

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

auto bdoc = document{} <<
    "_id" << bsoncxx::oid{ entityId.toStdString() } <<
    "moved" << spt::itest::update::doc( entityId ) <<
    finalize;
const auto reqv = bdoc.view();
auto payload = QByteArray( reinterpret_cast<const char*>( reqv.data() ), reqv.length
    () );

const auto endpoint = QString( "%1/%2" ).arg( url ).arg( entityId );
const auto reply = put( endpoint, payload, &req );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error updating document" );
const auto body = reply->readAll();
const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "document" ) != option->end(), "Update response did not
    return updated document" );
QVERIFY2( option->find( "history" ) != option->end(), "Update response did not return
     history metadata" );
```

```

const auto udoc = (*option)["document"].get_document().view();
QVERIFY2( udoc.find( "_id" ) != option->end(), "Update response did not return id" );
QVERIFY2( udoc.find( "moved" ) != option->end(), "Update did not add new moved
    property" );
QVERIFY2( udoc.find( "prop1" ) == option->end(), "Update did not remove old property"
     );
```

26.6 mergeById

26.6.1 C++

```
QString url{ "http://localhost:6106/crud/update/itest/test" };

const auto custom = []( const QString& url, const QByteArray& verb,
    QNetworkRequest* req, const QByteArray& data )
{
  QEventLoop eventLoop;
  connect( &mgr, &QNetworkAccessManager::finished, &eventLoop, &QEventLoop::quit );

  req->setUrl( QUrl( url ) );
  req->setAttribute( QNetworkRequest::Http2DirectAttribute, {true} );
  ReplyPointer rptr{ mgr.sendCustomRequest( *req, verb, data ) };
  eventLoop.exec();
  return rptr ;
};

using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::finalize;

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

auto bdoc = document{} <<
    "_id" << bsoncxx::oid{ entityId.toStdString() } <<
    "added2" << "yet another property" <<
    finalize;
const auto reqv = bdoc.view();
auto payload = QByteArray( reinterpret_cast<const char*>( reqv.data() ), reqv.length
    () );

const auto endpoint = QString( "%1/%2" ).arg( url ).arg( entityId );
const auto reply = custom( endpoint, "PATCH", &req, payload );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error patching document" );
const auto body = reply->readAll();
const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "document" ) != option->end(), "Patch response did not return
     updated document" );
QVERIFY2( option->find( "history" ) != option->end(), "Patch response did not return
    history metadata" );
```

```cpp

const auto udoc = (*option)["document"].get_document().view();
QVERIFY2( udoc.find( "_id" ) != option->end(), "Patch response did not return id" );
QVERIFY2( udoc.find( "moved" ) != option->end(), "Patch removed old property" );
QVERIFY2( udoc.find( "added2" ) != option->end(), "Patch did not add property" );
```

26.7 replace

26.7.1 C++

```cpp
QString url{ "http://localhost:6106/crud/replace/itest/test" };

using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::open_document;
using bsoncxx::builder::stream::close_document;
using bsoncxx::builder::stream::finalize;

auto doc = document{} <<
    "filter" << open_document << "_id" << bsoncxx::oid{ entityId.toStdString() } <<
    close_document <<
    "replace" << open_document << "bson" << "value" << close_document <<
    finalize;

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

const auto reply = post( url,
    { reinterpret_cast<const char*>( doc.view().data() ), static_cast<int>( doc.view
    ().length() ) },
    &req, "application/bson" );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error creating document" );
const auto body = reply->readAll();
const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "document" ) != option->end(), "Replace response did not
    return replaced document" );
QVERIFY2( option->find( "history" ) != option->end(), "Replace response did not
    return version history metadata" );
```

Chapter 27

CRUD

27.1 retrieve

27.1.1 C++

```cpp
QString url{ "http://localhost:6106/crud/retrieve/itest/test" };

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

const auto endpoint = QString( "%1/prop1/value1" ).arg( url );
const auto reply = get( endpoint, &req );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error retrieving api response" )
    ;
const auto body = reply->readAll();

const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "results" ) != option->end(), "results not returned" );

const auto arr = (*option)["results"].get_array().value;
QVERIFY2( !arr.empty(), "results array empty" );
const auto obj = arr[0].get_document().value;
QVERIFY2( obj.find( "_id" ) != obj.end(), "id not returned" );
QVERIFY2( obj.find( "prop1" ) != obj.end(), "prop1 not returned" );
QVERIFY2( obj.find( "prop2" ) != obj.end(), "prop2 not returned" );
QVERIFY2( obj.find( "prop3" ) != obj.end(), "prop3 not returned" );
QVERIFY2( obj.find( "prop4" ) != obj.end(), "prop4 not returned" );
```

27.2 query

27.2.1 C++

```cpp
QString url{ "http://localhost:6106/crud/query/itest/test" };

using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::open_document;
using bsoncxx::builder::stream::close_document;
using bsoncxx::builder::stream::finalize;

QNetworkRequest req;
req.setRawHeader( "accept", "application/bson" );

auto doc = document{} <<
  "query" << open_document << "prop1" << "value1" << close_document <<
  "options" <<
    open_document <<
      "projection" << open_document << "prop1" << 1 << close_document <<
    close_document <<
  finalize;
const auto reply = post( url,
  { reinterpret_cast<const char*>( doc.view().data() ), static_cast<int>( doc.view().
    length() ) },
  &req, "application/bson" );

QVERIFY2( reply->error() == QNetworkReply::NoError, "Error retrieving api response" )
    ;
const auto body = reply->readAll();

const auto option = bsoncxx::validate( reinterpret_cast<const uint8_t*>( body.data()
    ), body.size() );
QVERIFY2( option.has_value(), "Response not BSON" );
QVERIFY2( option->find( "results" ) != option->end(), "results not returned" );

const auto arr = (*option)["results"].get_array().value;
const auto obj = arr[0].get_document().value;
QVERIFY2( obj.find( "_id" ) != obj.end(), "id not returned" );
QVERIFY2( obj.find( "prop1" ) != obj.end(), "prop1 not returned" );
QVERIFY2( obj.find( "prop2" ) == obj.end(), "prop2 returned" );
QVERIFY2( obj.find( "prop3" ) == obj.end(), "prop3 returned" );
QVERIFY2( obj.find( "prop4" ) == obj.end(), "prop4 returned" );
```

List of Tables